SENGA ： 紙上の線画ビジュアルプログラミング言語

SENGA: 紙上の線画ビジュアルプログラミング言語

SENGA: LINE ART VISUAL PROGRAMMING LANGUAGE ON PAPER

秋山草太
Sota AKIYAMA
指導教員　宮本健司

法政大学大学院理工学研究科応用情報工学専攻修士課程

Trick Reality is a technique that makes real things seems as if they are moving in reality. In Trick Reality, a high-level programming language for editing line art is introduced, and makes it possible for line ars to be seen as if transforming. In the conventional method, programs are described in ordinary text style on system side,which prohibits using the functionality on end-user side.. In this research, we propose a line-art-style programming language written on paper in the same way as the contents that manipulates the edited contents of line art. In this method, a function to edit line art has been added to the "line art". By using this method, the line art itself actually written has a function to edit the line art, so that the descriptive power of end-user's line art is extended.

Keywords *: Trick Reality , Line Art , Graph , Visual Programming*

１．はじめに

紙上の線画を操作するための、同じく紙上の線画で構成されるビジュアルプログラミング言語の報告を行う.

トリックリアリティー[1][2]は現実世界の線画をさも現実空間で動いているかのようにアニメーションさせる手法である． 我妻[1]は輪郭を利用した手書き図形の分解を行い、線画を編集する高級言語を作成した. 柳川[2]は輪郭情報からグラフ構造を抽出しベクトル化を行い現実世界の線画の動的な変形を行った.

これまでの技術では現実世界の紙面上に描かれるのは対象となる線画のみであり、線画をどう編集させるかの記述にはコンピューター上で高級言語でのプログラミングを行うことで編集を指定していた． この結果、エンドユーザすなわち線画を記述する側の表現力は限定されていた。

本研究では、紙に書かれた線画に対するプログラムを同様に紙に線画で描く言語を提案する． 提案する言語では線画のデータに対して線の追加、削除の機能を提供する． 本方式でプログラムを書くための記述はすべて線画で書ける図形であり、線画を書く延長としてプログラムを書くことが出来る． 本方式を用いることで線画への編集を線画として現実に書くことが出来るので、線画の書き手の表現力が拡張される。.

本方式は線画に線画を編集するための意味を持たせたものである. また本方式を用いると、線画の線の数に応じて別の線画の線を減らすようなプログラムを書くことが可能となる.

２．線画グラフ

線画とは、線を組み合わせ描く図形のことである.

線画をグラフ構造と解釈し、同じ線画を復元可能とする線画グラフについて述べる.

（１）定義

線画の線を**エッジ**呼び、線画の端点・分岐点を**ノード**と呼ぶ. また線画で囲われた領域を**エリア**、線画を囲う領域を**アウターエリア**と呼ぶ. エッジのうちアウターエリアに接するものを特に**アウターエッジ**と呼ぶ. エッジとアウターエッジの集合 E、ノードの集合 N とするとき線画グラフは以下で示される.

エッジにノードを対応させる写像

$$f : E \to power(N) \tag{1}$$

ここで power(N)は N の冪集合をあらわす.

ノード n に接続したエッジを時計回りで表す列を n の**結節辺環**と呼ぶ。ノードに結節辺環を対応させる写像

$$g : N \to q(E) \tag{2}$$

の組

$$S : = (E, N, f, g) \tag{3}$$

を**線画グラフ**と定義する.

線画グラフにおいて閉路のうち囲むエリアが単一であるものを**最小閉路**という

（２）同型判定

２つの線画グラフ構造が同じことを線画グラフとして同型と呼ぶ。また一方の線画グラフのエッジをエッジの連なりにマッチすることを許し、グラフ構造が同じ部分をもつことを線画グラフとして部分同型であると呼ぶ。

図 1 に部分同型の例を示す図 2，3 に同型ではない例を示す.

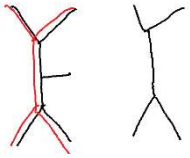


図１ 部分同型１

図２ 内外

図３　鏡像

図は中心の辺を２つの辺の連なりにマッチさせることで部分同型となる赤い線画が部分同型の部分となっている.図２の内外および図３鏡像は線画グラフとしてアウターエッジと結節辺環の構成が異なるので同型ではない.

３．SENGA：紙上の線画ビジュアルプログラミング言語

SENGA では現実世界の紙やホワイトボードなどに手書きで書いた線画をカメラでコンピューターに取り込みその線画を解析し、プログラムを実行する.

実行結果は取り込んだ線画を編集した線画となる.実行した結果の線画をスクリーン上に映し出し、さも現実の線画が編集されたかのように表示する.この章では SENGA の記法、システム構成およびその解析方法を述べる.

（１）システム構成

システム構成を図 4 に示す.

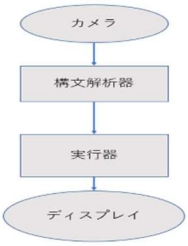


図４　システム構成

現実世界に手書きの線画で本言語を書く．カメラを利用して書いた線画を画像として PC に取得する. PC で線画をグラフ化して解析することで本言語として理解する．また理解したプログラムを PC 上で実行する.

カメラを用いて現実世界の線画を取り込み、構文解析器を用いて、ビジュアルプログラミング言語を理解する.理解した言語を用いて実行器で現実世界の線画を編集し、ディスプレイに表示することでプログラムに応じた現実世界の線画への編集を実現する.

（２）トークン

本言語は以下のトークンを持つ.

処理矢印：３分岐によってつながった３エッジ

演算矢印：２分岐によってつながった２エッジ

サークル：アローに属さない最小閉路とその内部領域

特殊サークル：アローに属さない最小閉路を辺共有し４つ繋げたもの

コンテント：サークルの内部領域に書かれた線画グラフ

表 2 にトークンを示す.

表１　トークン一覧

トークン名	記述例	説明
サークル		アローに属さないエッジで囲まれた領域
特殊サークル		サークルの内部に接しながら×を書いた形
処理矢印		辺が３つの矢印
演算矢印		辺が２つの矢印
コンテント	自由に記述可能	サークルの内部に書く

特に特殊サークルにコンテントを書くことはできない.

（３）構文論

サークルの集合 *C*、処理矢印の集合 P、演算矢印の集合 O とする.

開始点となる処理矢印がただ一つのみ存在し、**開始処理矢印**と呼ぶ.開始処理矢印の矢じりにはただ一つの円が接続する.開始処理矢印以外の矢印に矢じり、矢羽根の両端点に同一サークルを許し、ただ一つのサークルが接続する.

また各矢印から考えて矢羽根側に接続されたサークルを**元サークル**、矢じり側に接続されたサークルを**先サークル**と呼ぶ.

本構文により矢印とサークルからなる最小閉路が生成されることがあるが、構文解析アルゴリズムによってトークンを判別する.詳細は 4.2 構文解析アルゴリズムの項を参照

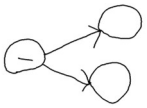
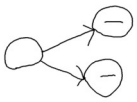
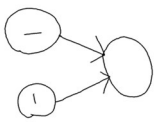
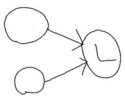
（４）意味論

a）処理矢印

元サークルのコンテントをコピーし、先サークルのコンテントとする.のちに元サークルのコンテントを初期値に戻す.一つのサークルを元サークルとし二つ以上の処理矢印が接続されていた場合、処理は並列に行われる.反対に一つのサークルを先サークルとし二つ以上の処理矢印が接続されていた場合、それぞれのコンテントが合成される.処理矢印は元サークルにコンテントが渡って来たとき動作する.元サークルのコンテントが負である場合その処理矢印は動作しない.

動きの例を表 2 に示す.

表 2　処理矢印の動作

動作前	動作後	説明
		矢羽根側のコンテントが矢じり側に移る
		分岐していた場合、それぞれにコンテントが移る
		合流していた場合、それぞれのコンテントが合成される

b）演算矢印

元サークルのコンテントを決定し、先サークルのコンテントとする.コンテントの決定法は以下である

元サークルにコンテントがあるか

　ある：そのコンテント

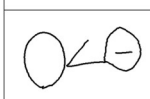
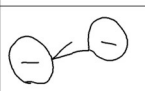
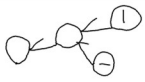
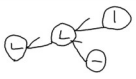
　ない：その元サークルを先サークルとする演算矢印 x があるか

　　ある：全ての x の元サークルの決定したコンテントを合成したコンテント

　　ない：空のコンテント

表 3 に演算矢印の動作の例を示す.

表 3 演算矢印の動作.

動作前	動作後	説明
		矢羽側のコンテントが矢じり側にコピーされる
		矢羽根側にさらに演算矢印が接続されているとその処理も行う

c）サークル

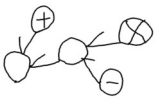
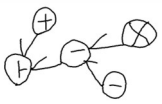
コンテントを保持する.

現実にコンテントを持っている場合にはそのコンテントで固定される.

処理矢印によってコンテントが送られてきたとき、自分が先サークルとなっている演算矢印の処理を行う.

特殊サークルは×―1 という意味を持ち、合成が発生したとき合成する対象を負にする.負のコンテントと合成が発生したとき対象のコンテントからそのコンテントを取り除くという処理になる.表 4 に特殊サークルの動作例を示す

表 4　特殊サークルの動作

動作前	動作後	説明
		線が一つ無くなる

（５）利用例

図 5 にプログラムの記述例を示す.

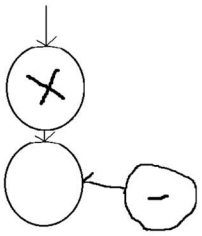


図 5.本プログラム記述例

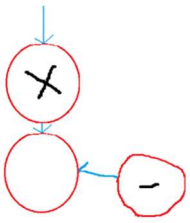


図 6.記述例をトークンごとに色分けした例

2 種類の矢印と円を繋げて書くことでプログラムを記述する. 図 5 に図 6 のプログラムをトークンごとに色分けしたものを示す.サークルを赤、アローを青、コンテントを黒で色分けした.図 7 に図 5 のプログラムを実行した結果を示す.

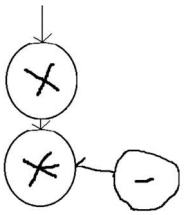


図 7 記述例の実行結果.

実行されるとサークルに値が表示される.開始処理矢印の先サークルから次演算矢印の先サークルへコンテントが渡され、次のサークルに接続された演算矢印の元サークルのコンテントと合成された結果が先サークルに描画されている.

図 8.に実際に本ビジュアルプログラミング言語を現実の紙面で書いている様子を示す. 図 9.に書いた言語をコンピューターに取り込み実行している様子を示す.

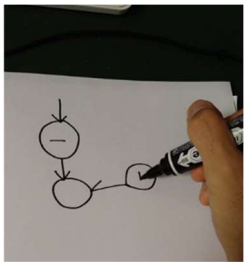


図 8. 実際に SENGA を書いている様子.

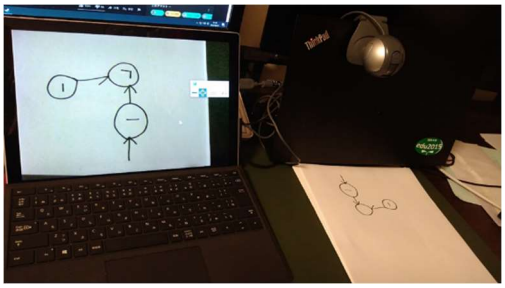


図 9. 本プログラムを実行している様子.

左がコンピューターおよびディスプレイ、右下に書いた線が、右上にカメラがある.書いたプログラムは縦線に横線を合成するというものである.結果、実行結果を表示するディスプレイには、処理矢印の先のサークルに縦と横の線画がくっついた線画がさも紙に書かれたように書かれている.

４． アルゴリズム

（１）線画グラフの同形、部分同形判定

同型、部分同型を検出するアルゴリズムを疑似コードで示す.SameTree 関数を呼び出すことで同型判定が行える.ループまたは先端を検知すると次の親に進むようにして作る木を Tree 関数によって作成し、その木と同様な木が作成できるか全ての構成を試す関数となっている.同じ木が作成できるということは同じ線画グラフが作成できるということであり、部分グラフであると言える.

```
//roop[n]: ノード n が閉路であったとき true
//done[n]: ノード n について探索済みのエッジ集合
//child[n]: ノード n の子の列
for n in 全てのノード do
   child[n] ← {}
   done[n] ← {}
   loop[n] ← false
end do

function SubTree(n,e,stk)//n：ノード　e: 親からのエッジ
stk:親スタック
if n in stk then do
   done[n]に e を追加
   loop[n] ← true
   n ← 新規ノード
   child[n] ← {}
   done[n] ← {}
   loop[n] ← true
   return n
end do
s ←　n の結節辺環
stk に n を追加
for ne in s from e do
   if ne is e then do
      continue
   end do
   else if ne in done[n] then do
      continue
   end do
   else then
      done[n]に ne を追加
      nn ← n から ne を進んだ先のノード
      child[n]に SubTree(nn,ne,stk)の結果を追加
   end do
end do
stk から n を除去
return n

function Tree(n,e)//n:ノード　e:アウターエッジ
if n is Φ then do //線画が円であるとき
   n ← 新規ノード
   child[n] ← {}
   done[n] ← {}
   loop[n] ← true
   nn ← 新規ノード
   child[nn] ← {}
   done[nn] ← {}
   loop[nn] ← true
```

```
    child[n]に nn を追加
  end do
  else if e is  Φ  then do//点があるとき
    n ← 新規ノード
    child[n] ← {}
    done[n] ← {}
    loop[n] ← false
  end do
  else do
    stk ← {n}
    s ←  n の結節辺環
    for ne in s from e do
      if ne in done[n] then do
        continue
      end do
      else do
        done[n]に ne を追加
        nn ← n から ne を進んだ先のノード
        child[n]に SubTree(nn,ne,stk)の結果を追加
      end do
    end do
  end do
  return n


  function SameTree(ga,gb)
  //ga:線画グラフ  gb:ga の部分グラフか調べる線画グラフ
  be ← gb のアウターエッジのうちどれか
  bn ← be に対応したノードのうちどちらか
  n ← Tree(bn,be)
  for ae in ga のアウターエッジ  do
    for an in ae に対応したノード  do
      stk ← {}
      ret ← SubST(ae,an,be,bn,stk)
      if ret is Φ then do
        continue
      end do
      else do
        return ret
      else do
   end do
  return Φ

  function SubST(ae,an,be,bn,stk)
  s ←  an の結節辺環
  if child[bn] is {} then do
    if loop[bn] is true then do
      if an in stk then do //ループ終り
        done[an]に ae を追加
        loop[an] ← true
    n ← 新規ノード
    child[n] ← {}
    done[n] ← {}
    loop[n] ← false
    return n
  end do
  else do//先にループ終りがないか探す
    for ne in s from ae do
      if ne is ae then do
        continue
      end do
      else do
        nn ← an から ne を進んだ先のノード
        stk に an を追加
        ret ← SubST(ne,nn,be,bn,stk)
        stk から an を除去
        if ret is Φ then do
          continue
        end do
        else do
          done[an]に ae を追加
          child[an]に ret を追加
          return an
        end do
      end do
    end do
      return Φ
    end do
    else do//子が無くてループでない　常に OK
      done[an]に ae を追加
      return an
    end do
  else do
    if loop[bn] is true then do//子に渡してはいけない
      cbn ← child[bn]の先頭
      for ne in s from ae do
        if ne is ae then do
          continue
        end do
        else do
          nn ← an から ne を進んだ先のノード
          stk に an を追加
          ret ← SubST(ne,nn,be,bn,stk)
          stk から an を除去
          if ret is Φ then do
            continue
          end do
        else do
          done[an]に ne を追加
          child[an]に ret を追加
```

```
      cbn ← child[bn]の cbn の次
      if cbn is Φ then do
        return an
      end do
      else do
        continue
      end do
    end do
  end do
  return Φ
end do
else do
  cbn ← child[bn]の先頭
  for ne in s from ae do
    if ne is ae then do
      continue
    end do
    else do
      nn ← an から ne を進んだ先のノード
      stk に an を追加
      ret ← SubST(ne,nn,be,bn,stk)
      stk から an を除去
      if ret is Φ then do
        continue
      end do
      else do
        done[an]に ne を追加
        child[an]に ret を追加
        cbn ← child[bn]の cbn の次
        if cbn is Φ then do
          return an
        end do
        else do
          continue
        end do
      end do
    end do
//自分の子では出来ないので子に投げる
    for ne in s from ae do
      if ne is ae then do
        continue
      end do
      else do
nn ← an から ne を進んだ先のノード
stk に an を追加
ret ← SubST(ne,nn,be,bn,stk)
stk から an を除去
if ret is Φ then do
  continue
end do
        else do
          done[an]に ne を追加
          child[an]に ret を追加
          return Φ
        end do
      end do
    end do
//子でもできないので不可
    return Φ
  end do
end do
```

（２）線画グラフの合成

a）正の線画グラフ A と正の線画グラフ B の合成

両元サークルのコンテントの適当なノードを選び、そのノードが重なるように描画する.その描画されたものをもう一度グラフと解析することで合成した結果のグラフを得る.

b）負の線画グラフ A と負の線画グラフ B の合成

正と正と同じ処理を行い、符号を負とする.

c）正の線画グラフ A と負の線画グラフ B の合成

エッジ数が少ないグラフがもう一方の部分グラフかどうかを確認し、部分グラフであればその部分を除去する.部分グラフでなければ適当に少ないほうのエッジの数だけ多いほうのエッジを除去する.符号はエッジ数の多い符号と同じにする.

d）3 個以上のグラフの合成

符号が同じものを 2 個ずつ合成し、その後符号が異なる合成を行う.

（３）構文解析

図.をコンピューターに取り込んだ時、最も外側の領域として認識されるのは図. において黄色く示された範囲である. この領域を領域 A とする. 領域 A に接している線画をグラフ化したときに得られるノードが図 9 で赤く示した点である. このグラフを作成したときノードの数が 4 以上であれば本言語である可能性がある. 異なるときさらに他や内側の領域の線画のグラフのノード数を確認していく.本言語である可能性があるとき開始処理矢印の捜索を行う.開始処理矢印は先ノードのみ接続された処理矢印である.したがって接続エッジ数が５のノードのうち 1 ホップ先のノードの接続エッジ数が１であるものが 3 つであるノードが開始処理矢印の矢じりノードとなり接続エッジ数が１であるノードが各羽ノードとなる.図. で得られたノードを接続エッジ数で色分けしたものを図. に示す.上記の様に図.の青いノードから１ホップ先に赤いノードが３つある紫に塗ったエッジ群が開始処理矢印になることが分かる.

開始処理矢印の矢じりノードの接続エッジに接している領域のうち開始処理矢印に接していない領域が開始処理矢印の先サークルとなる領域となり、その領域を囲っ

ているエッジ、ノード群が先サークルを構成することとなる.このノードに接続エッジ数が４のノードがあるとき特殊サークルであるので追加で接続エッジ数が４のノードに接続されているエッジそれぞれに接している領域が囲っているエッジ、ノードをサークルを構成するものとして追加する.

以上の様にして開始処理矢印と一番目のサークルを決定した後以下の処理を繰り返すことで全てのサークル、アロー、その接続関係を決定する.

処理済みでないサークルを調べる.そのサークルを構成するノードの接続エッジのうちノードを構成するエッジ以外をそれぞれ E とする.E がすでにアローとして処理済みかを調べ、未処理であれば両端ノードの接続エッジ数が[3,4]か[3,5]かによってアローの種類を決定し､さらに処理中サークル側のノードの接続エッジ数によって元ノード、先ノードどちらかを埋める.その後 E の反対のノードの接続エッジに接している領域のうち E に接していない領域が次のサークルとなる領域である.その領域、領域を囲っているエッジ、ノードを次のサークルとしてノードを構成する.このノードに接続エッジ数が４のノードがあるとき特殊サークルであるので追加で接続エッジ数が４のノードに接続されているエッジそれぞれに接している領域を囲っているエッジ、ノードをサークルを構成するものとして追加する.そして矢印の先サークルか元サークルとして登録する.ここで作成したサークルを未処理のサークルとして登録する.

以上の処理を繰り返すことで全てのサークル、アロー、その接続関係を得ることが出来る.

（４）実行

実行開始時、開始処理矢印の先サークルが次のアクティブとして登録される.

実行を行うと次のアクティブとして登録されたサークルが今アクティブなサークルとなり各今アクティブなサークルについて以下の処理が行わる.

１．　値の決定

自サークルに値があるときその値.ないとき自サークルを先サークルとする各演算矢印の元サークルの値と該当ノードに渡された値をすべて合成した値.

２．　決定したサークルの値の表示

３．　該当サークルの元サークルする処理矢印の先サークルを次のアクティブとして登録する該当サークルの値が空か正であるとき先サークルに該当サークルの値を渡す.

今アクティブなサークルをひとまとめとして考え、全ての処理が終わることを１ステップと呼ぶ.

1 ステップを次々繰り返すことにより本言語は実行されていく.

今アクティブなサークルの遷移例を図 10.に示す

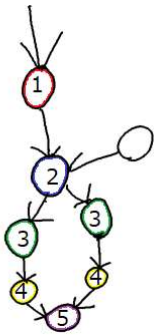


図 10. アクティブなサークルの遷移.

1 から５まで順に遷移する.初め開始処理矢印の先サークルである 1 と書かれたサークルが今アクティブなる.次に１から伸びた処理矢印の先サークルである２と書かれたサークルが今アクティブとなる.その時接続された演算矢印はアクティブとあは関係なく無視される.次に２から伸びた 2 本の処理矢印の先サークルである３と書かれたサークル２つが同時に今アクティブとなる.次に 2 つのアクティブなサークルそれぞれについて伸びた処理矢印の先サークルである４と書かれたサークル 2 つが同時に今アクティブとなる.次に 2 つのアクティブなサークルから伸びた処理矢印の先サークルである合流した 1 つの５と書かれたサークルが今アクティブとなる.このように動作が起こるアクティブなサークルが遷移してゆくことでプログラムが順に実行されていく.

５．実装

実装は C++を用いて行った.ライブラリとして描画に OpenGL、画像解析に DirectX を使用した.

描画の実装では各ノードの中心座標を求めコンテントの位置を所属ノードの中心からの相対位置で座標を保持する.コンテントがサークルを移動する際には所属サークルが Model 行列を作成しコンテントはそのモデル行列を利用すればコンテントの実際の位置にかかわらずサークルの位置に描画が可能となる.

取り込んだ線画を編集するため線画グラフを新たなグラフへと変換した.そのグラフは任意のエッジの除去、導入が可能である.エッジを除去した際にエッジに接続されたノードの接続エッジ数が０になるとそのノードも除去される.あるエッジを導入する際には互いのノードの位置を合わせ描画を行う.またエッジの除去、導入後は領域の構成等のずれが生じる可能性があるので、変更した線画を描画しもう一度線画グラフを構築しなおすことで正しい線画グラフを得る.

図 11.に本言語で記述可能な複雑なプログラムを示す.

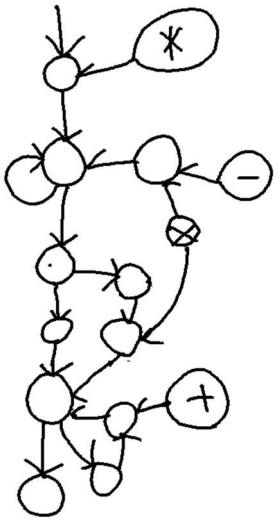


図 11. 複雑なプログラム.

複雑なプログラムで描いたプログラムは大きく二つに分けることが出来る. コンテントでドットをもつサークルの前と後で役割が大きく変わる. 前ではコンテントにエッジが８本の米印が入り、次のサークルでエッジ１の負のコンテントを合成し、自己ループと次に進んでいる. 処理矢印はコンテントが負でないときのみ動作するので８回ループし、８回次に進む処理矢印が動作することになる.

ドット後ではドットが正と負になり合成され空になる. 空のコンテントに対して十字のコンテントが合成され、そのコンテントが次に合成されるコンテントとなる. またドット前の処理より８回動作が行われることとなるので. 十字が８回合成された形が最終的に出力されることになる. 図. に複雑なプログラムの実行結果を示す.

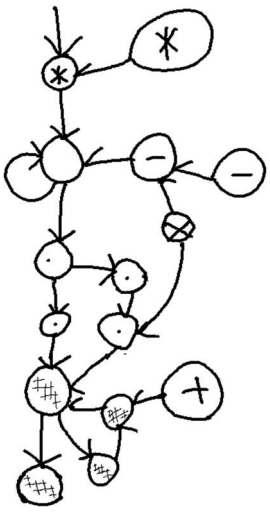


図 12.　複雑なプログラムの実行結果.

６．結論

本研究では線画グラフを利用することにより現実の紙に記述し解析可能でかつ現実に書いた線画のグラフ構造を変更することができるビジュアルプログラミング言語を提案した.

線画とその線画を編集するプログラムを同じ紙面上に記述できるようになることで、今まで現実に書いた線画に対して編集を行う場合にはコンピューターに取り込み、コンピューター上でプログラミングを行わなければならなかったが、その一連の作業を現実の紙面上のみで可能となった. また線画とプログラムをすべて現実の紙に書くことが出来るので、線画に対する編集を行うプログラミングがより直感的となり、書きやすくなったといえる.

本研究で提案した線画グラフの同形判定アルゴリズムにより人間が見たときの線画において同じと考えられる部分を同形だと判定することが可能になったと考えられる.

本方式では線画を解析することで頂点・辺を取得し、その構成を利用して言語を実現している.　したがって線画として解析できない太い線では利用できない.　またカメラで画像を取得する特性上、暗い場所では利用することが出来ない.

本研究では編集可能なデータとして線画のみを扱ったがデータの解析に物体認識を利用すればより適応範囲が増えると考えられる.

謝辞: 指導教官の宮本健司准教授には、研究へのアドバイスや論文の構成にいたるまで様々なご指導をただきました. ここに感謝いたします.

参考文献

1) [1]我妻利彦 (2015)「輪郭を利用した手書き図形の分解」法政大学理工学部応用情報工学科 2015 年度卒業論文.
2) [2]柳川裕樹 (2017)「グラフ構造抽出・ベクトル化による線画の変形」法政大学理工学部応用情報工学科 2017 年度卒業論文..